



MODUL PRAKTIKUM

# PEMROGRAMAN BERBASIS FRAMEWORK

## Modul 9

- Global API service (GET)
- Global API Service (POST)
- Global API Service (DELETE)

JURUSAN TEKNOLOGI INFORMASI  
POLITEKNIK NEGERI MALANG

# Global API

Pada Modul 4 yang dulu kita pelajari mengenai bagaimana kita berinteraksi dengan API. Kita dihadapkan akan bagaimana cara kita mengelola banyak daftar alamat API yang disediakan dan mengumpulkannya menjadi satu kelompok. Sehingga kita dapat menambah, mengurangi, ataupun memperbaiki daftar API pada satu tempat.

Salah satu caranya adalah dengan mengelola/mengatur pemanggilan API secara GLOBAL, sehingga kita tidak perlu melakukan “*coding ulang*” API yang memiliki karakter yang sama. Contoh seperti saat kita memanggil API (GET) untuk mendapatkan *list* Artikel, *list* komentar, dan *list* pengguna. API untuk ketiga proses tersebut sebenarnya sama, yang membedakan hanya alamat url API yang dituju, sehingga kita bisa menaruh list API pada satu tempat untuk mempermudah dalam mengelolanya.

Pada praktikum pembahasan Global API ini, kita menggunakan bahan dari materi yang sebelumnya yaitu Modul 4 – Interaksi dengan API. Sehingga kita dapat memahami perbedaan antara Modul 4 yang hanya melakukan interaksi dengan API dengan modul ini (Modul 8) yang mengelola daftar list API kedalam satu tempat sehingga proses manajemen API lebih mudah dan cepat.

Keuntungan menggunakan Global API Bagi Para Developer Antara Lain:

1. Terpusat.

API yang dibuat atau yang akan dipanggil terpusat pada satu tempat.

2. Kustomisasi

Mudah dan efisien saat melakukan kustomisasi maupun perbaikan alamat API.

3. Mudah di manage

Berhubung list API berada pada satu tempat, sehingga proses manage daftar API baik penambahan API, pengurangan, maupun memperbaiki API (perubahan alamat url API) bisa dilakukan dengan mudah.

4. Terhindar dari proses yang sama (redundansi)

Berhubung daftar API dalam satu tempat sehingga kita bisa memantau meminimalisir proses yang sama atau pembuatan API yang sama persis.

# Praktikum 1

## Global API service GET

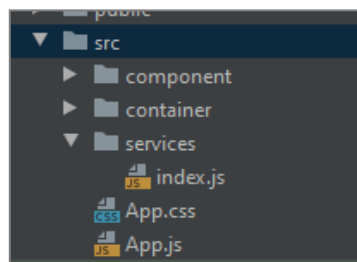
### 1.1 Run Project dan Server Fake API

Sebelum memulai praktikum, kita mulai terlebih dahulu menyiapkan source hasil dari praktikum Modul 4 dan menjalankan server project dan server Fake API tersebut, yaitu buka 2 jendela *command prompt* (CMD) pada project yang sudah kita buat (pada Modul 4). Jendela pertama kita ketikkan perintah `npm start` untuk menjalankan local server project, dan jendela CMD kedua kita isikan perintah `json-server --watch listArtikel.json --port 3001` untuk menjalankan server fake API.

### 1.2 Langkah Praktikum

Dalam pembuatan Global Service API ini kita akan memerlukan tempat untuk mengumpulkan resource API yang ada, maka kita akan membuat tempat untuk menampung resource tersebut

1. Buat folder baru bernama “**Services**” dalam folder **src**, kemudian buat file **index.js** seperti pada Gambar 1.1



Gambar 1.1. Pembuatan folder dan file untuk Global Service API

2. Buka file **BlogPost.jsx** pada folder container (*statefull component*) dan akan tampil kode program seperti Gambar 1.2.

```

1 import React, {Component} from "react";
2 import './BlogPost.css';
3 import Post from "../../component/BlogPost/Post";
4
5 class BlogPost extends Component{
6   state = { // komponen state dari React untuk statefull component
7     listArtikel: [], // variabel array yang digunakan untuk menyimpan data API
8     insertArtikel: { // variabel yang digunakan untuk menampung sementara data yang akan di insert
9       userId: 1, // kolom userId, id, title, dan body sama, mengikuti kolom yang ada pada listArtikel.json
10      id: 1,
11      title: "",
12      body: ""
13    }
14  }
15
16  ambilDataDariServerAPI = () => { // fungsi untuk mengambil data dari API dengan penambahan sort dan order
17    fetch( input: 'http://localhost:3001/posts? sort=id& order=desc') // penambahan sort dan order berdasarkan parameter
18      .then(response => response.json()) // ubah response data dari URL API menjadi sebuah data json
19      .then(jsonHasilAmbilDariAPI => { // data json hasil ambil dari API kita masukkan ke dalam listArtikel pada state
20        this.setState( state: {
21          listArtikel: jsonHasilAmbilDariAPI
22        })
23      })
24  }
25
26  componentDidMount() { // komponen untuk mengecek ketika component telah di-mount-ing, maka panggil API
27    this.ambilDataDariServerAPI() // ambil data dari server API lokal
28  }

```

Gambar 1.2. Isi kode program BlogPost

3. Pada fungsi `ambilDataDariServerAPI` (baris16) terdapat pemanggilan API GET untuk me-request data artikel. Proses dari fungsi inilah yang akan kita manage kedalam satu tempat yaitu `index.js`.

4. Buka file `index.js` pada folder `service`, yang telah kita buat tadi dan tuliskan baris kode seperti

Gambar 1.3 berikut

```

1 const domainPath = 'http://localhost:3001'; // simpan url domain server API pada variabel, sehingga bisa dinamis (diganti)
2 const GetAPI = (path) => { // path digunakan untuk menunjuk alamat API mana yang akan di-request
3   const promise = new Promise( executor, (resolve, reject) => {
4     fetch( input: `${domainPath}/${path}`) // alamat url domain + path untuk mengakses full alamat API yg di-request
5       .then(response => response.json()) // response dari server harus dijadikan json
6       .then((result) => {
7         resolve(result); // jika success menerima response dari server maka resolve response ke user
8       }, (err) => {
9         reject(err); // jika terjadi error dari server (server down, dll),
10        // maka kirim pesan error ke user melalui reject.
11      })
12   });
13   return promise;
14 }
15
16 const getNewsBlog = () => GetAPI( path: 'posts? sort=id& order=desc');
17
18 const API = { // inisialisasi function-function yang akan disediakan global API.
19   getNewsBlog
20 }
21
22 export default API;

```

Gambar 1.3. kode program index.js

5. Kembali ke file `BlogPost.js`. Ganti baris 17 sampai baris 23 menjadi seperti Gambar 1.4

```
16      ambilDataDariServerAPI = () => {                                // fungsi untuk mengambil data dari API dengan penambahan sort dan order
17          API.getNewsBlog().then(result => {
18              this.setState( state: {
19                  listArtikel: result
20              })
21          })
22      }
```

Gambar 1.4. edit kode program BlogPost

6. Simpan file `BlogPost.js` dan `index.js` tersebut, dan amati apa yang terjadi pada browser kalian.

### 1.3 Pertanyaan Praktikum 1

- a. Perhatikan file `index.js`, apa tujuan dibuatnya fungsi `GetAPI` pada baris 2 dan fungsi `getNewsBlog` pada baris 16?

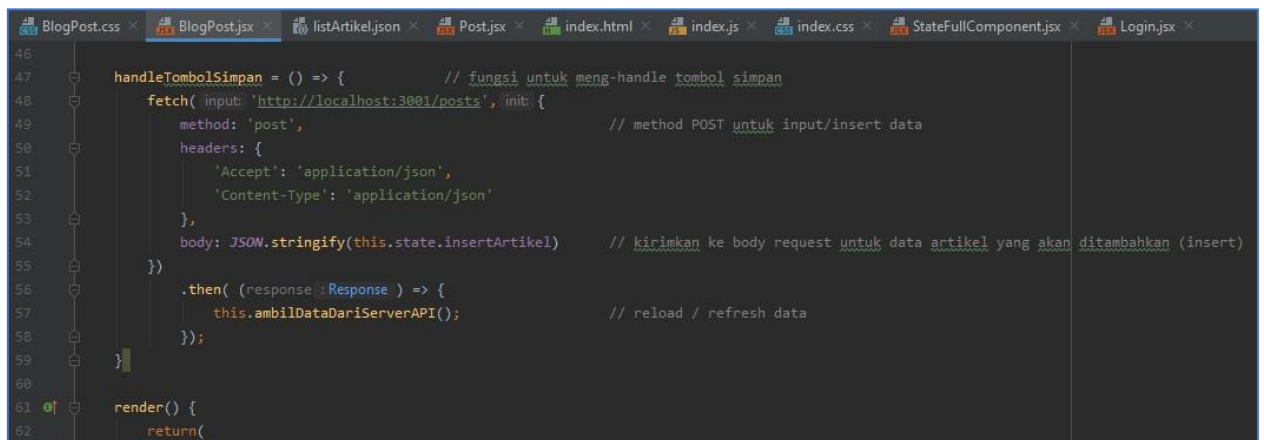
## Praktikum 2

### Global API service POST

#### 2.1 Langkah Praktikum 2

Seperti pada langkah praktikum 1 dimana kita *me-manage* API GET untuk mendapatkan data dari server. Sekarang kita akan *me-manage* API POST untuk mengirimkan data kepada server API. Langkah-langkahnya adalah

1. Kita perhatikan pada fungsi `handleTombolSimpan` pada file `BlogPost.js`. Bagian inilah yang selanjutnya kita kelola agar bisa di-manage.



```
46
47 handleTombolSimpan = () => { // fungsi untuk meng-handle tombol simpan
48   fetch( input: 'http://localhost:3001/posts', init: {
49     method: 'post', // method POST untuk input/insert data
50     headers: {
51       'Accept': 'application/json',
52       'Content-Type': 'application/json'
53     },
54     body: JSON.stringify(this.state.insertArtikel) // kirimkan ke body request untuk data artikel yang akan ditambahkan (insert)
55   })
56   .then( (response : Response ) => {
57     this.ambilDataDariServerAPI(); // reload / refresh data
58   });
59 }
60
61 render() {
62   return(
```

Gambar 2.1. Kode program BlogPost fungsi handleTombolSimpan

2. Buatlah fungsi untuk menampung action POST dari file `BlogPost.js` pada file `services/index.js` dan **inisialisasi** fungsi tersebut seperti pada Gambar 2.2

```

15 const PostAPI = (path, data) => {
16   const promise = new Promise( executor: (resolve, reject) => {
17     fetch( input: `${domainPath}/${path}`, init: {
18       method: 'post', // method POST untuk input/insert data
19       headers: {
20         'Accept': 'application/json',
21         'Content-Type': 'application/json'
22       },
23       body: JSON.stringify(data) // kirimkan ke body request untuk data artikel yang akan ditambahkan (insert)
24     })
25     .then((result :Response) => {
26       resolve(result); // jika success menerima response dari server maka resolve response ke user
27     }, (err) => {
28       reject(err); // jika terjadi error dari server (server down, dll),
29     })
30   })
31   return promise;
32 }
33
34 const getNewsBlog = () => GetAPI( path: 'posts?_sort=id&_order=desc');
35 const postNewsBlog = (dataYangDiKirim) => PostAPI( path: 'posts', dataYangDiKirim);
36
37 const API = { // inisialisasi function-function yang akan disediakan global API.
38   getNewsBlog,
39   postNewsBlog
40 }
41
42 export default API;

```

Gambar 2.2. Kode program index.js untuk handle POST

- Selanjutnya pindah ke file `BlogPost.js` dan ganti isi dari fungsi `handleTombolSimpan` menjadi seperti Gambar 2.3

```

handleTombolSimpan = () => { // fungsi untuk meng-handle tombol simpan
  API.postNewsBlog(this.state.insertArtikel)
  .then( (response) => {
    this.ambilDataDariServerAPI(); // reload / refresh data
  });
}

```

Gambar 2.3. Kode program pengganti handleTombolSimpan

- Selesai. Kode program pada `BlogPost.js` sedikit lebih simple dari seelumnya.
- Silahkan kalian jalankan proses input artikel melalui browser dan amati apa yang terjadi.

## 2.2 Pertanyaan Praktikum 2

- Perhatikan file `index.js`, apa tujuan dibuatnya fungsi `PostAPI` dan fungsi `postNewsBlog`?
- Pada fungsi `postNewsBlog`, terdapat variable `dataYangDiKirim`. Apa tujuan dari dibuatnya variable tersebut?

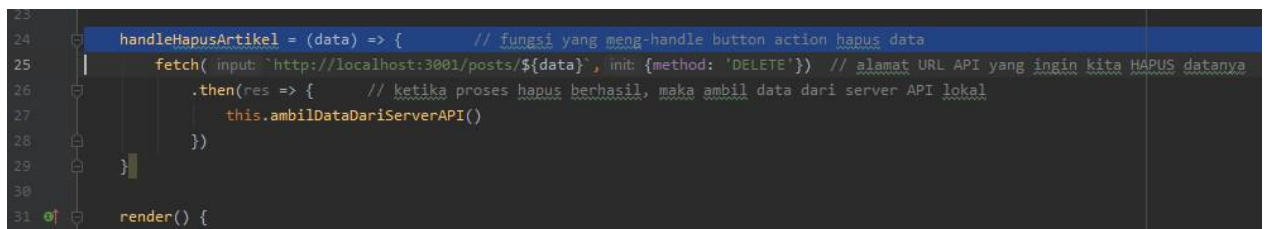
## Praktikum 3

### Global API service DELETE

#### 3.1 Langkah Praktikum 3

Seperti pada langkah praktikum 2 dimana kita *me-manage* API GET dan POST untuk mendapatkan data dari server. Sekarang kita akan *me-manage* API DELETE untuk request hapus data pada server API. Langkah-langkahnya adalah

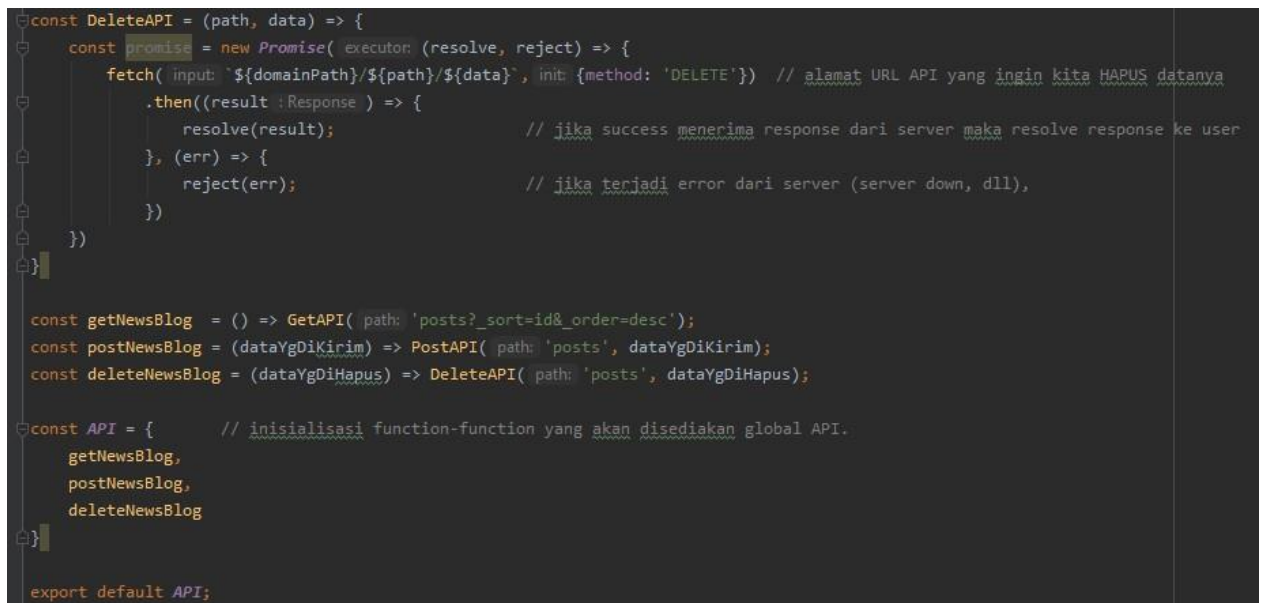
1. Kita perhatikan pada fungsi `handleHapusArtikel` pada file `BlogPost.jsx`. Bagian inilah yang selanjutnya kita kelola agar bisa di-manage.



```
23
24 handleHapusArtikel = (data) => { // fungsi yang meng-handle button action hapus data
25   fetch( input: 'http://localhost:3001/posts/${data}', init: {method: 'DELETE'}) // alamat URL API yang ingin kita HAPUS datanya
26   .then(res => { // ketika proses hapus berhasil, maka ambil data dari server API lokal
27     this.ambilDataDariServerAPI()
28   })
29 }
30
31 render() {
```

Gambar 3.1. Kode program BlogPost fungsi handleHapusArtikel

2. Buatlah fungsi untuk menampung action DELETE dari file `BlogPost.jsx` pada file `services/index.js` dan **inisialisasi** fungsi tersebut seperti pada Gambar 3.2



```
const DeleteAPI = (path, data) => {
  const promise = new Promise( executor: (resolve, reject) => {
    fetch( input: `${domainPath}/${path}/${data}`, init: {method: 'DELETE'}) // alamat URL API yang ingin kita HAPUS datanya
    .then((result :Response ) => {
      resolve(result); // jika success menerima response dari server maka resolve response ke user
    }, (err) => {
      reject(err); // jika terjadi error dari server (server down, dll),
    })
  })
}

const getNewsBlog = () => GetAPI( path: 'posts?_sort=id&_order=desc');
const postNewsBlog = (dataYgDiKirim) => PostAPI( path: 'posts', dataYgDiKirim);
const deleteNewsBlog = (dataYgDiHapus) => DeleteAPI( path: 'posts', dataYgDiHapus);

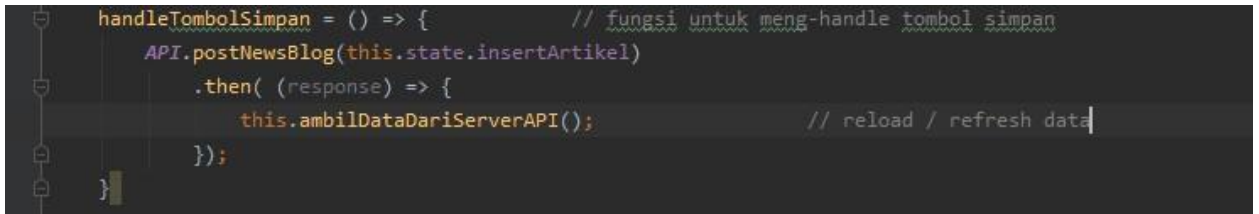
const API = { // inisialisasi function-function yang akan disediakan global API.
  getNewsBlog,
  postNewsBlog,
  deleteNewsBlog
}

export default API;
```

Gambar 3.2. Kode program index.js untuk handle DELETE



3. Selanjutnya pindah ke file `BlogPost.jsx` dan ganti isi dari fungsi `handleHapusArtikel` menjadi seperti Gambar 3.3



```
handleTombolSimpan = () => { // fungsi untuk meng-handle tombol simpan
  API.postNewsBlog(this.state.insertArtikel)
    .then( (response) => {
      this.ambilDataDariServerAPI(); // reload / refresh data
    });
}
```

Gambar 3.3. Kode program pengganti `handleTombolSimpan`

4. Selesai. Kode program pada `BlogPost.jsx` sedikit lebih simple dari sebelumnya.
5. Silahkan kalian jalankan proses hapus artikel melalui browser dan amati apa yang terjadi.

### 3.2 Pertanyaan Praktikum 3

1. Perhatikan file `index.js`, apa tujuan dibuatnya fungsi `DeleteAPI` dan fungsi `deleteNewsBlog`?
2. Pada fungsi `deleteNewsBlog`, terdapat variable `dataYangDiHapus`. Apa tujuan dari dibuatnya variable tersebut?

## Praktikum 4

### Manage Global API service

#### 4.1 Manage Global API

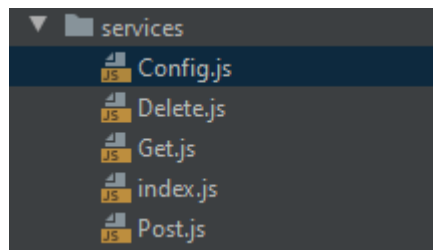
Pada global API yang telah kita lakukan pada praktikum 1, 2 dan 3. Kita mengetahui bahwa saat kita benar-benar melakukan coding untuk membuat API, kita akan dihadapkan pada banyak url API yang akan kita sediakan. Sehingga kita butuh manage lagi Global API untuk lebih teratur.

Salah satu cara untuk *me-manage* Global API adalah dengan memisahkan API berdasarkan action-nya (GET, POST, DELETE). Sehingga, missal saat terjadi error pada DELETE API, kita cukup akan membuka Global API DELETE yang akan kita perbaiki.

#### 4.2 Langkah Praktikum 4

Langkah-langkah yang dilakukan pada praktikum 4 ini adalah

1. Buatlah file `Config.js`, `Get.js`, `Post.js`, dan `Delete.js` dalam folder `services` seperti pada Gambar 4.1



Gambar 4.1. Buat file

2. Buka `Config.js` dan isikan kode seperti Gambar 4.2

```
index.js x Post.js x Delete.js x Config.js x Get.js x BlogPost.js x
1 // file config ini bisa berisi variabel-variabel lain yang dibutuhkan untuk proses API
2 export const domainPath = 'http://localhost:3001'; // simpan url domain server API pada variabel, sehingga bisa dinamis (diganti)
```

Gambar 4.2. Kode program config.js

3. Buka `Get.js` dan isikan kode seperti Gambar 4.3

```

import {domainPath} from './Config';

const GetAPI = (path) => {
  // path digunakan untuk menunjuk alamat API mana yang akan di-request
  const promise = new Promise( executor: (resolve, reject) => {
    fetch( input: `${domainPath}/${path}` ) // alamat url domain + path untuk mengakses full alamat API yg di-request
      .then(response => response.json()) // response dari server harus dijadikan json
      .then((result) => {
        resolve(result); // jika success menerima response dari server maka resolve response ke user
      }, (err) => {
        reject(err); // jika terjadi error dari server (server down, dll),
      }) // maka kirim pesan error ke user melalui reject.
    })
  return promise;
}
export default GetAPI;

```

Gambar 4.3. Kode program Get.js

4. Buka **Post.js** dan isikan kode seperti Gambar 4.4

```

import {domainPath} from './Config';

const PostAPI = (path, data) => {
  const promise = new Promise( executor: (resolve, reject) => {
    fetch( input: `${domainPath}/${path}`, init: {
      method: 'post', // method POST untuk input/insert data
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data) // kirimkan ke body request untuk data artikel yang akan ditambahkan (insert)
    })
      .then((result :Response ) => {
        resolve(result); // jika success menerima response dari server maka resolve response ke user
      }, (err) => {
        reject(err); // jika terjadi error dari server (server down, dll),
      })
    })
  return promise;
}
export default PostAPI;

```

Gambar 4.4. Kode program Post.js

5. Buka **Delete.js** dan isikan kode seperti Gambar 4.5

```

import {domainPath} from './Config';

const DeleteAPI = (path, data) => {
  const promise = new Promise( executor: (resolve, reject) => {
    fetch( input: `${domainPath}/${path}/${data}`, init: {method: 'DELETE'}) // alamat URL API yang ingin kita HAPUS datanya
      .then((result :Response ) => {
        resolve(result); // jika success menerima response dari server maka resolve response ke user
      }, (err) => {
        reject(err); // jika terjadi error dari server (server down, dll),
      })
    })
  return promise;
}
export default DeleteAPI;

```

Gambar 4.5. Kode program Delete.js

6. Pada index.js kita edit menjadi seperti Gambar 4.6

```
import GetAPI from "../Get";
import PostAPI from "../Post";
import DeleteAPI from "../Delete";

// Daftar API - GET
const getNewsBlog = () => GetAPI( path: 'posts?_sort=id&_order=desc');

// Daftar API - POST
const postNewsBlog = (dataYgDiKirim) => PostAPI( path: 'posts', dataYgDiKirim);

// Daftar API - DELETE
const deleteNewsBlog = (dataYgDiHapus) => DeleteAPI( path: 'posts', dataYgDiHapus);

const API = { // inisialisasi function-function yang akan disediakan global API.
  getNewsBlog,
  postNewsBlog,
  deleteNewsBlog
}

export default API;
```

Gambar 4.6. Kode program index.js

7. Amati apa yang terjadi

### 4.3 Pertanyaan Praktikum 4

1. Bagaimana caranya untuk menambahkan daftar API baru baik untuk method GET, POST, DELETE pada Global API?

## **TUGAS PRAKTIKUM**

Ubahlah **Tugas pada Modul 4** yang sudah kalian buat dengan memanfaatkan Global API seperti praktikum yang kita lakukan pada Modul 8 ini.

~ ~ ~ **Selamat Mengerjakan** ~ ~ ~